



</>

Intro to Python Programming



- Python: a beginner-friendly, powerful language for automation, data, web, and creative coding



- Ideal for beginners, students, and creators—write less code, achieve more



- Course flow: syntax → program flow → functions → a small working program



- Succeed with hands-on practice, curiosity, and a growth mindset



```
def greet(name):  
    message = f"Hello, {name}!"  
    return message  
  
name = "Python Learner"  
print(greet(name))
```

Setting Up Your Python Environment

-  Python 3.14.6 is the latest stable release as of July 2026
-  Installation is straightforward on Windows, macOS, and Linux
-  Choose a beginner-friendly editor: Thonny, VS Code, or IDLE
-  Write your first line:
`print('Hello, world!')`
-  The Python interpreter reads and executes your code line by line



Python Syntax Fundamentals



- - Variables store data like strings, integers, floats, and booleans



- - Use snake_case for readable variable and function names



- - Add single-line comments with # to explain your code



- - Consistent indentation defines code blocks, replacing braces



- - Watch for missing colons, mismatched quotes, and indentation errors

```
1  # Example: Python syntax fundamentals
2
3  name = "Alex"
4  age = 25
5  height = 5.9
6  is_student = True
7
8  # This function greets the user
9  def greet_user(user_name):
10     greeting = f"Hello, {user_name}!"
11     return greeting
12
13  # Call the function
14  message = greet_user(name)
15  print(message)
16
```

Variables store different types of data

Comments explain your code

Indentation defines the code block

Function call and output

Controlling Program Flow



- Use `if`, `elif`, and `else` to make decisions in code



- Compare values with `==`, `!=`, `and`, `or`, `not` operators



- Repeat actions with `for` loops over lists and ranges

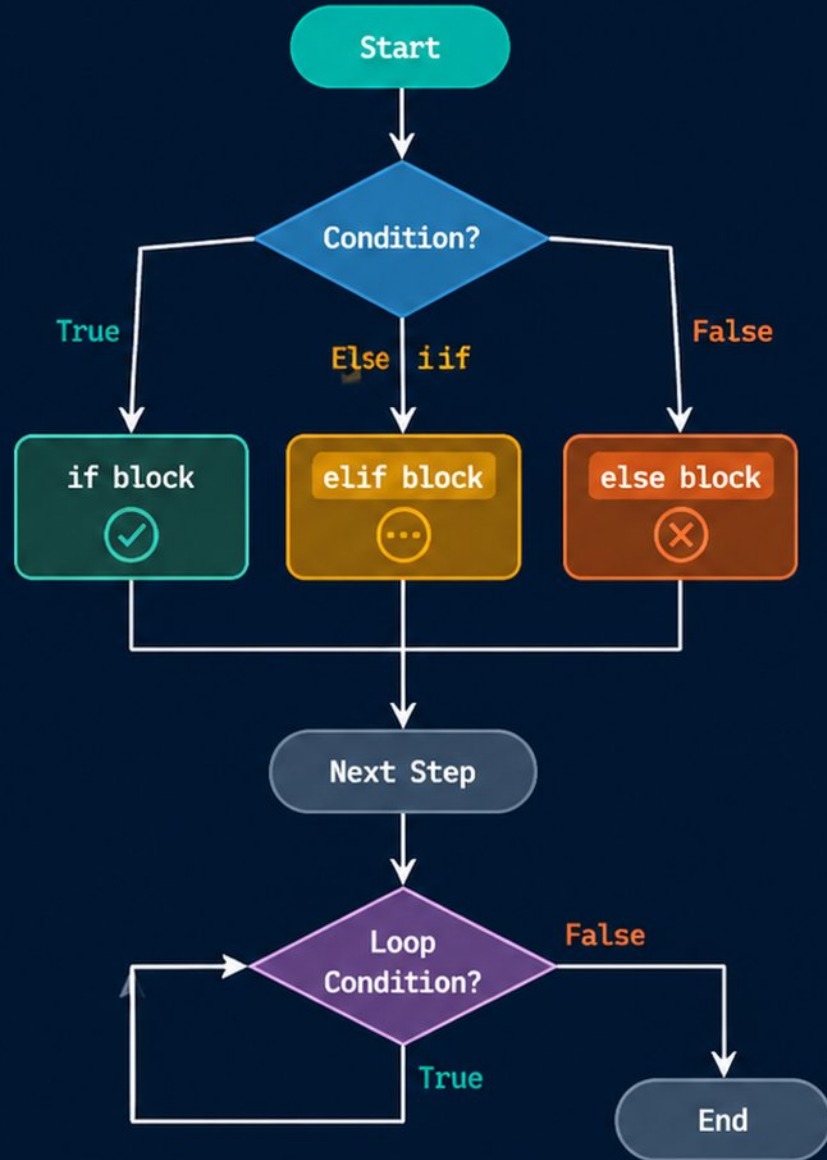


- Use `while` loops carefully to avoid infinite executions



- Sketch flowcharts to visualize branching logic and loops

```
{  
  _____  
  _____  
  _____  
}
```



Working with Data Collections

LISTS

Ordered, mutable collection



```
shopping_list = ["Milk", "Bread",  
                 "Apples", "Coffee"]
```

TUPLES

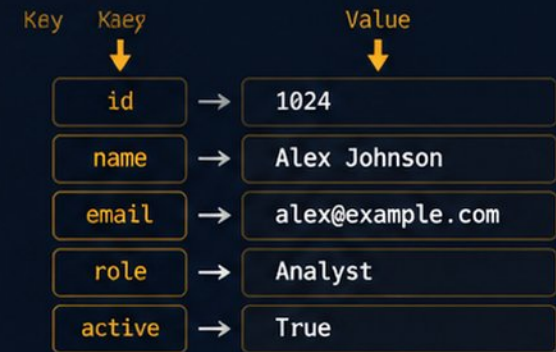
Immutable sequence



```
coordinates = (37.7749, -122.4194,  
               15.0)
```

DICTIONARIES

Key-value pairs



```
user = {"id": 1024, "name": "Alex Johnson",  
        "email": "alex@example.com",  
        "role": "Analyst", "active": True}
```

- **Lists**: ordered, mutable collections for storing sequences
- **Tuples**: immutable sequences used for fixed data
- **Dictionaries**: key-value pairs for structured information
- **Key methods**: `.append()`, `.remove()`, and `.sort()` for lists
- **Simple examples**: shopping list, coordinates, and user profiles



Writing and Using Functions



def()

- Define functions with `def` and call them by name



- Parameters receive data; arguments are the values you pass in

return

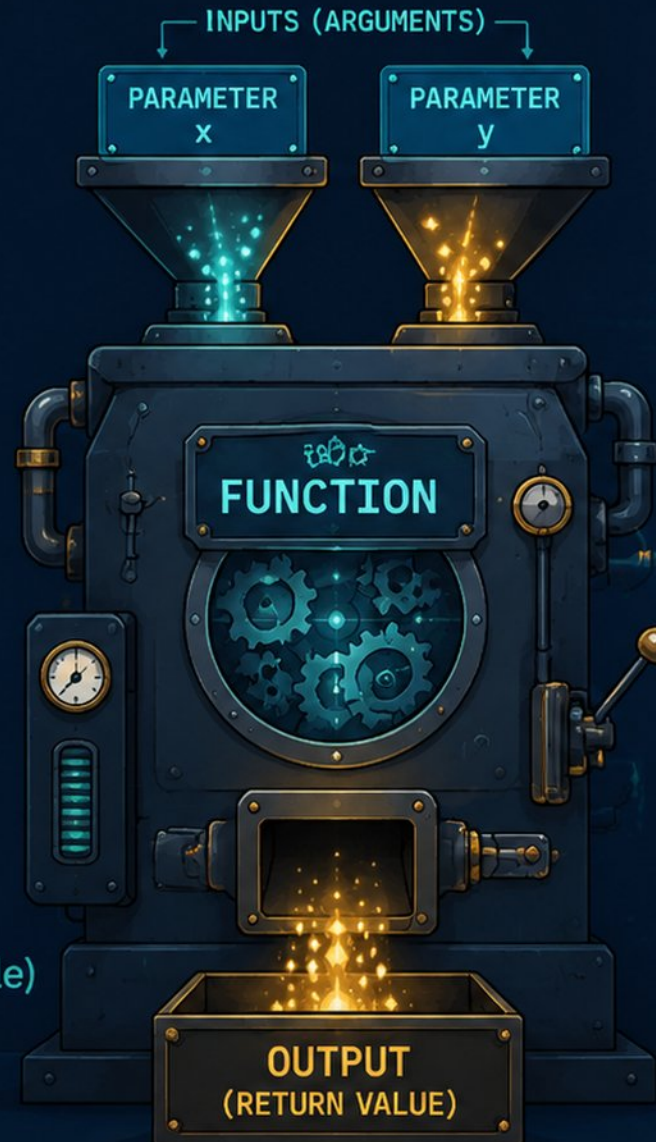
- Use `return` to send a result back, otherwise get `None`



- Local variables stay inside functions, globals are accessible everywhere



- Write functions to avoid repeating code (DRY principle)



FUNCTION DEFINITION

```
def add(x, y):  
    result = x + y  
    return result
```

Parameters (receive data) Return value (sent back)



Reading and Writing Files

- Open files with `open()` using 'r', 'w', 'a' modes
- Use `with` statements for automatic, safe file closing
- Read data via `.read()`, `.readline()`, or `.readlines()`
- Write to files and handle basic errors gracefully
- Practical uses: saving notes, reading CSVs, logging data



>_ File Modes

'r' = read (default)
'w' = write (overwrite)
'a' = append (add to end)

```
1 with open('data.txt', 'r') as f:
2     content = f.read()
3     for line in f.readlines():
4         print(line)
5 # File is closed automatically
6 # and safely
```



Building Your First Small Program



- **Project overview:** a simple to-do list app with task management



- **Combine** functions, loops, conditionals, and file I/O into one program



- **Step-by-step walkthrough:** add, view, complete, and save tasks



- **Debugging tips:** print statements, reading error messages, testing small chunks



- **Next steps:** practice on PyChallenge, ZenPy, PYNative, and Genepy



Simple To-Do List App

```
1 import json
2
3 tasks = []
4
5 def add_task(task):
6     tasks.append({"task": task, "done": False})
7     print("Task added.")
8
9 def view_tasks():
10     for i, t in enumerate(tasks, 1):
11         status = "✓" if t["done"] else "X"
12         print(f"{i}. [{status}] {t['task']}")
13
14 def complete_task(index):
15     if 1 <= index <= len(tasks):
16         tasks[index - 1]["done"] = True
17         print("Task completed.")
18
19 def save_tasks(filename="tasks.json"):
20     with open(filename, "w") as f:
21         json.dump(tasks, f, indent=2)
22         print("Tasks saved.")
```

Write, Test & Debug Your Program

NEXT STEPS



PyChallenge



ZenPy



PYNative



Genepy

Keep Practicing & Level Up



PersonWise.

PersonWise • AI digital-human course platform



Scan the code or click anywhere to watch this course live, with voice Q&A

personwise.ai/t/3e64eb5d/public